¿Con cual de los siguientes comandos de Maven no se ejecutarían las pruebas unitarias?

Ax mvn clean compile surefire:test

B mvn test-compile surefire:test

C mvn clean test-compile surefire:fire

D mvn tets

Al ejecutar una clase de test que contiene n drivers:

A el método anotado con @beforeeach se ejecuta n.-1 veces

B el método anotado con @afterall se ejecuta n veces

Cx el método anotado con @beforeall se ejecuta una vez

D el método anotado con @aftereach se ejecuta n +1 veces

Si al aplicar el método de caja negra de particiones equivalentes, obtenemos las siguientes particiones de entrada, válidas y no válidas, teniendo en cuenta la siguiente codificación para identificar las particiones: 'E' denota entrada; 'V' denota válida; 'nV' denota no válida:

Entrada 1: E1V1, E1nV1
Entrada 2: E2V1, E2nV1, E2nV2

Indica cuál es la cardinalidad del conjunto de casos de prueba eficiente y efectivo obtenido al aplicar dicho método

A 4

B 5

C No se puede obtener si no se conocen las particiones de salida válidas y no válidas

D

Con el método de camino básico de McCabe

A Debemos elegir el conjunto mímo de caminos para conseguir que todas las sentencias se ejecuten al menos una vez en cada caso de prueba

B todas las opciones son falsas

C debemos elegir el conjunto mínimo de caminos para conseguir ejecutar todas las condiciones al menos una vez en cada caso de prueba

D Debemos elegir todos los caminos del grafo

Para realizar pruebas de una SUT que contiene dependencias externas

a) Usaremos una verifacion basada en el estado para pruebas unitarias y una verificación basada en el comportamiento para pruebas de integración
b) Todas son falsas
c) El primer paso es identificar los seams que contiene la SUT
d) No está permitido modificar la SUT de ninguna forma para realizar pruebas unitarias

Indica la línea o líneas en las que tenemos puntos de inyección de seams para la SUT calculaConsumo():

```
1.//paquete ppss.ejercicio2
2.public class GestorLlamadas {
3.  static double TARIFA_NOCTURNA=10.5;
4.  static double TARIFA_DIURNA=20.8;
5.
6.  public Calendario getCalendario() {
7.    Calendario c = new Calendario();
8.    return c;
9.  }
10.
11.  public double calculaConsumo(int minutos) {
12.    Calendario c = getCalendario();
13.    int hora = c.getHoraActual();
14.    if(hora < 8 || hora > 20) {
15.      return minutos * TARIFA_NOCTURNA;
16.    } else {
17.       return minutos * TARIFA_DIURNA;
18.    }
19.  }
20.}
```

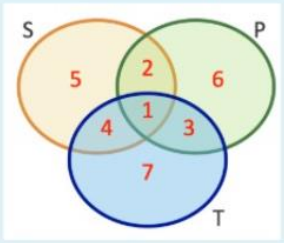
Seleccione una:

- las líneas 12 y 13
- la línea 6
- Dejo la pregunta en blanco
- no hay ningún punto de inyección
- la línea 7

Utilizando el método de caja negra de particiones equivalentes, si tenemos una entrada asociada a un tipo enumerado con 3 valores, indica cual de als siguientes afirmaciones es falsa

a) Podemos tener una sola partición valida de dicha entrada
b) Podemos tener tres particiones validas de dicha entrada
c) Podemos tener tres particiones no validas de dicha entrada
d) Podemos tener dos particiones validas de dicha entrada

Dado el siguiente diagrama de Venn que hemos trabajado en clase:



Indica cuál de las siguientes afirmaciones es cierta:

Seleccione una:

- Dejo la pregunta en blanco
- Un tester debe intentar que el subconjunto 7 sea lo más grande posible
- **Con métodos de caja negra y de caja blanca, se pueden alcanzar comportamientos de los subconjuntos 1 y 2**
- Con un método de caja negra se pueden alcanzar comportamientos del subconjunto 3
- Con un método de caja blanca se pueden alcanzar comportamientos del subconjunto 4

El artefacto Maven con las siguientes coordenadas:

ppss.examen:ejemplo:war:1.0-SNAPSHOT

El artefacto maven con las siguientes coordenadas:

`ppss.examen:ejemplo:war:1.0-SNAPSHOT`

representa el fichero:

Seleccione una:

- **$HOME/.m2/repository/ppss/examen/ejemplo/1.0-SNAPSHOT/ejemplo-1.0-SNAPSHOT.war**
- $HOME/.m2/repository/ppss/examen/ejemplo-1.0-SNAPSHOT.war
- $HOME/.m2/repository/ppss/examen/ejemplo/ejemplo-1.0-SNAPSHOT.war
- $HOME/.m2/repository/ppss/examen/ejemplo/war/1.0-SNAPSHOT/ejemplo-1.0-SNAPSHOT.war
- Dejo la pregunta en blanco

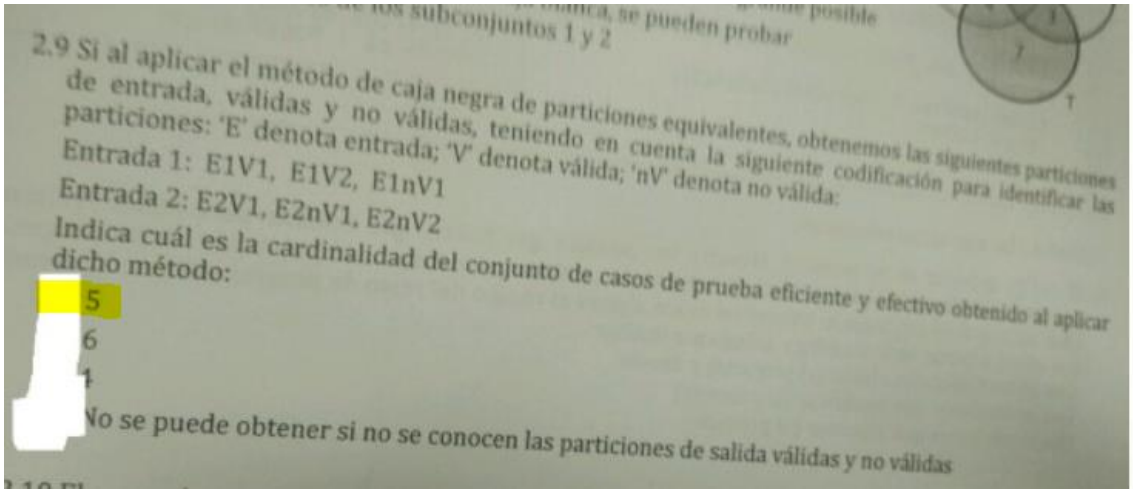
De los siguientes comandos, ¿Cuáles no ejecutarán los test unitarios?

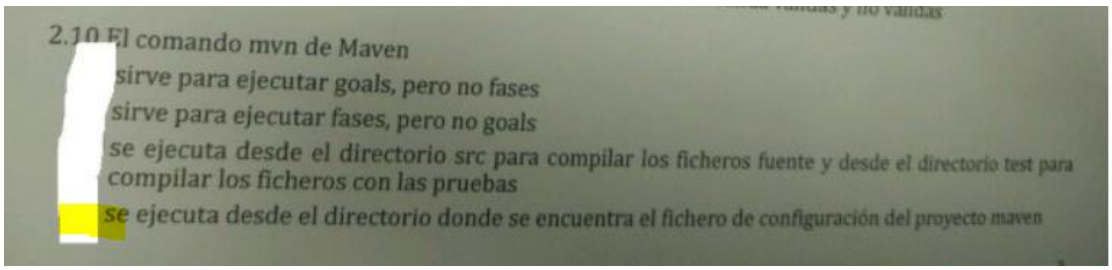
De los siguientes comandos de maven, ¿cuáles no ejecutarán los tests unitarios?

Comando 1: mvn clean surefire:test
Comando 2: mvn clean test
Comando 3: mvn test
Comando 4: mvn clean compile surefire:test
Comando 5: mvn clean test-compile surefire:test

Seleccione una:

- los comandos 2 y 3
- los comandos 1, 2, 4 y 5
- los comandos 1, 4 y 5
- Dejo la pregunta en blanco
- los comandos 1 y 4





Indica cual de las siguientes afirmaciones es falsa

a) Aunque probemos un código utilizando un conjunto de pruebas eficiente y efectivo y todos los test pasen, no podemos garantizar que el código presenta ningún defecto
b) Las pruebas pueden demostrar la ausencia de defectos
c) Durante el proceso de desarrollo de un producto, cuanto antes se detecte un defecto, menos costoso será repararlo
d) Aunque un producto funcione de acuerdo a los requerimientos especificados puede que no satisfaga las expectativas del cliente

Cualquier librería que sea requerida en el proceso de construcción de un proyecto Maven

a) Si se encuentra en un repositorio remoto, se descarga en el directorio target
b) Si no se encuentra en un repositorio remoto, se busca en el repositorio local
c) Si no se encuentra en un repositorio remoto, se busca en el directorio m2
d) Todas las afirmaciones son falsas

Indica cuál es la complejidad ciclomática del siguiente código:

```
if (a > b && a < c || c < b)
      a = b;
else if (a < b || c > a)
      c = a;
else
      b = c;
```

Seleccione una:

- 6
- 4
- 3
- Dejo la pregunta en blanco
- 5

Pregunta **22**

Sin responder aún

Puntúa como 0,39

Marcar pregunta

Si en el pom.xml de nuestro proyecto añadimos la siguiente propiedad:

```
<properties>
     <miPropiedad>misTests</miPropiedad>
</properties>
```

y la siguiente configuración del plugin maven-surefire-plugin:

```
<configuration>
           <groups>${miPropiedad}</groups>
 </configuration>
```

y tiene la siguiente clase para los tests con 3 drivers:

```
class MiClaseTest {
      @Tag("misTests")
      @Test void test1() {
            // aquí vendría el código del test
       }

      @Tag("otroTest")
      @Test void test2() {
            // aquí vendría el código del test
       }

      @Test void test3() {
            // aquí vendría el código del test
       }
}
```

Si desde línea de comandos ejecutamos la orden

```
mnv test
```

y tiene la siguiente clase para los tests con 3 drivers:

```
class MiClaseTest {
      @Tag(“misTests”)
      @Test void test1() {
            // aquí vendría el código del test
       }

      @Tag(“otroTest”)
      @Test void test2() {
            // aquí vendría el código del test
       }

      @Test void test3() {
            // aquí vendría el código del test
       }
}
```

Si desde línea de comandos ejecutamos la orden

```
mnv test
```

Seleccione una:

- se ejecutan los 3 drivers
- se ejecuta sólo test3()
- Dejo la pregunta en blanco
- se ejecuta sólo test1()
- no se ejecuta ningún driver porque en la orden no se indica ninguna etiqueta

Si en el pom.xml de nuestro proyecto añadimos la siguiente propiedad:

```
<properties>
     <miPropiedad>misTests</miPropiedad>
</properties>
```

y la siguiente configuración del plugin maven-surefire-plugin:

```
<configuration>
            <groups>${miPropiedad}</groups>
 </configuration>
```

y tiene la siguiente clase para los tests con 3 drivers:

```
class MiClaseTest {
      @Tag("misTests")
      @Test void test1() {
            // aquí vendría el código del test
       }

      @Tag("otroTest")
      @Test void test2() {
            // aquí vendría el código del test
       }

      @Test void test3() {
            // aquí vendría el código del test
       }
}
```

Si desde línea de comandos ejecutamos la orden

```
mvn test -DmiPropiedad=""
```

Seleccione una:

- se ejecuta sólo test1()
- se ejecuta sólo test3()
- Dejo la pregunta en blanco
- se ejecutan los 3 drivers
- no se ejecuta ningún driver porque en la orden no se indica ninguna etiqueta

Indica las líneas en las que identificamos las dependencias externas de la SUT realizaReserva():

```
//paquete ppss
1.  public class Reserva {
2.
3.     public boolean compruebaPermisos(String login, String password, Usuario tipoUsu) {
4.        throw new UnsupportedOperationException("Not yet implemented");
5.     }
6.
7.     public void realizaReserva(String login, String password,
8.                    String socio, String [] isbns) throws Exception {
9.
10.      ArrayList<String> errores = new ArrayList<>();
11.      if(!compruebaPermisos(login, password, Usuario.BIBLIOTECARIO)) {
12.         errores.add("ERROR de permisos");
13.      } else {
14.         IOperacionBO io = new Operacion();
15.        try {
16.           for(String isbn: isbns) {
17.              try {
18.                io.operacionReserva(socio, isbn);
19.              } catch (IsbnInvalidoException iie) {
20.                 errores.add("ISBN invalido" + ":" +  isbn);
21.              }
22.           }
23.        } catch (SocioInvalidoException sie) {
24.              errores.add("SOCIO invalido");
25.        } catch (JDBCException je) {
26.              errores.add("CONEXION invalida");
27.        }
28.     }
29.     if (errores.size() > 0) {
30.        String mensajeError = "";
31.        for(String error: errores) {
32.          mensajeError += error + "; ";
33.        }
34.        throw new ReservaException(mensajeError);
35.     }
36.  }
37. }
```

```
//paquete ppss
public enum Usuario {
  BIBLIOTECARIO, ALUMNO, PROFESOR
}
```

Seleccione una:

- ○ en las líneas 14 y 18
- ○ Dejo la pregunta en blanco
- ○ en las líneas 10 y 14
- ○ en las líneas 11 y 18